(1)

(15) N00014-75-C-0837

(6)

PERFORMANCE OF MINICOMPUTERS IN FINITE ELEMENT ANALYSIS PRE AND POST PROCESSING.

(10) (A)

J. Mai Tan H. Kamel
University of Arizona
Aerospace & Mechanical Engineering
Tucson, Arizona 85721

(12) 30 (11) 29 Jul 84

(11) 20 ~~May [redacted]80~~

(14) TR-6

(9) Technical Report No. 6
for period ending 20 May 80

Approved for public release, distribution unlimited

Department of the Navy
Office of Naval Research
Structural Mechanics Program (Code 474)
Arlington, Virginia 22217

402192 [illegible]

1. INTRODUCTION

The explosive rate of development of computer hardware has affected engineering practice, particularly the area of structural engineering and finite element analysis. Programs are now available, some on minicomputers, which perform pre- and postprocessing functions as well as analysis [1,2,3,4,5,6,7,8]. Many small engineering groups, enticed by the availability of relatively inexpensive hardware, have acquired mini- and microcomputer systems with the intention of using them for day to day engineering computations. This paper attempts a study of the performance of a typical 16-bit minicomputer in a finite element environment. It is hoped that this study will give a realistic feel for the performance of such systems in a practical environment, and provide an example of how the performance of such systems may be measured.

2. THE FINITE ELEMENT ENVIRONMENT

In order to measure the performance of the minicomputer in an environment representative of a finite element oriented engineering activity, a certain number of processes were selected as typical, and very specific combinations of these were chosen in order to perform the measurements. The following processes were considered typical:

a. Text editing : It is expected that a certain amount of text editing is necessary for engineering report writing, composition of letters, program development and so on.

b. Mesh Generation : The pre- and post processing activity is represented by a batch execution of a mesh generation program to produce the model shown in figure 1. The model has a total of 120 points, and 78 rectangular plate elements. It was generated using the BULKM mesh generation program, which is a part of the GIFTS-5 system [3].

c. Finite Element Computation: To represent this activity, a matrix decomposition was selected. Again the stiffness matrix for the problem shown in Figure 1 was chosen as the basis for a typical computation. Although the program used does not operate on a band width concept, it is still useful to note that the number of unknowns for the problem is 624. The maximum half band width is 120, and the computational (root mean square) half band width is 72. The program used, DECOM, is part of the GIFTS system. It uses a hyper-(partitioned) matrix generalization of the Cholesky decomposition algorithm.

d. FORTRAN Compilation and Program Linking: It was assumed that program development is an ongoing activity in certain engineering outfits, and that such development takes part side by side with finite element computations. For that purpose the compilation of a set of subroutines containing approximately 40 lines of coding each (Process FORT), and the linking of a main program with four subroutines (Process FORTLOAD), were included in some job profiles.

At this point it is necessary to explain the process which was selected for the performance measurement, as well as the limits that must be imposed on the findings. It is clear that

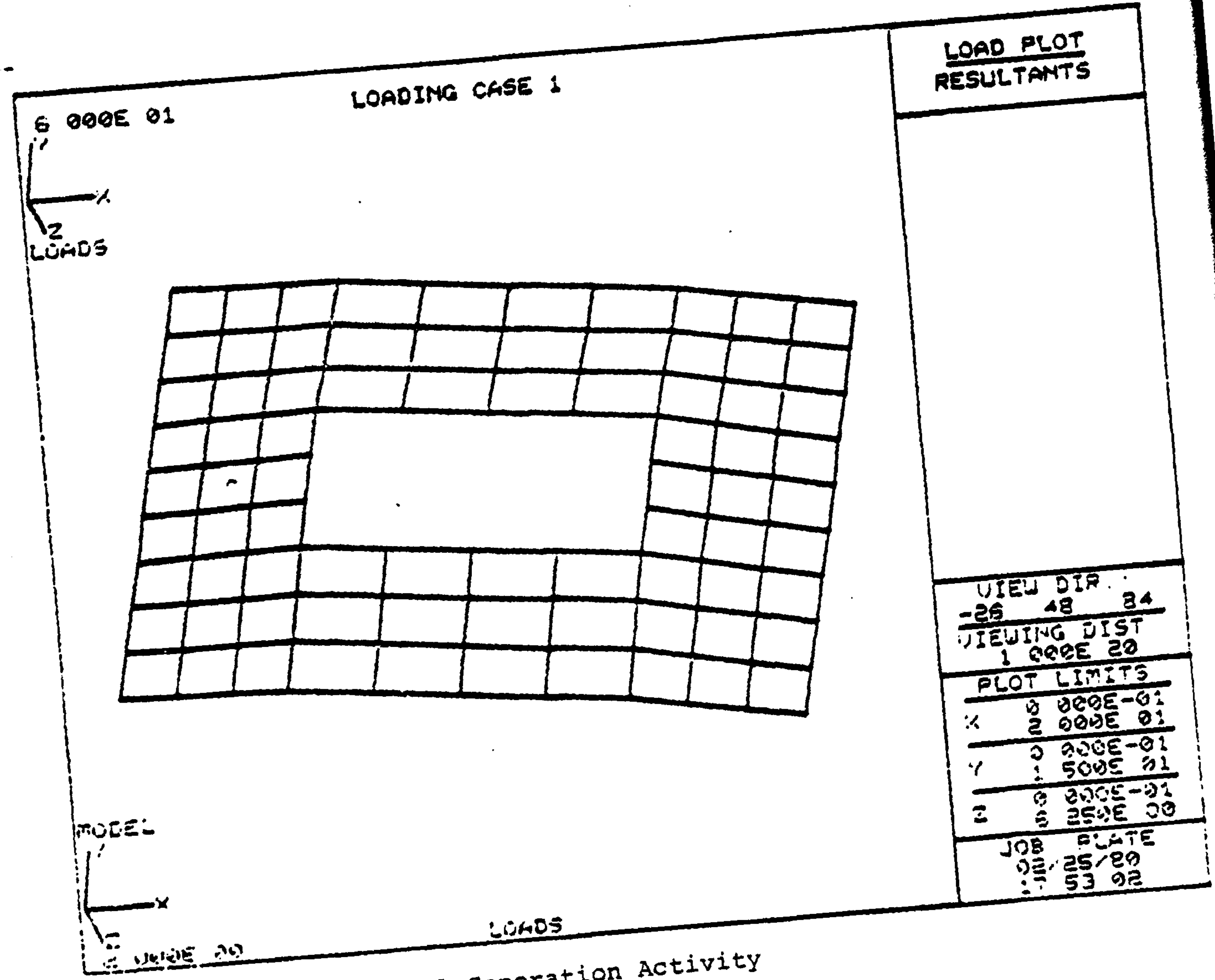

Figure 1. Model Generation Activity

the results obtained are dependent on the type of computer hardware used, the precise hardware configuration, the operating system involved, the number of users on the system and the exact nature of their activities. The performance also depends on many other extraneous factors, such as how full the disk is. Every effort was made, as will be described later in section 4, to reduce the amount of uncertainty and provide a predictable environment. In spite of this, it must be remembered that the data presented only address a specific computer system, FORTRAN compiler, linking editor, mesh generator and matrix decomposition program. The study should serve as a guide, as well as provide a basis for comparison with other computers, finite element programs and methods for benchmarking computer systems [9,10,11,12].

3. COMPUTER HARDWARE AND OPERATING SYSTEMS

Before proceeding further, it might be useful to introduce some terms, which will be helpful in discussing performance evaluation of FEM programs in minicomputers. A computer system can be divided into two logical parts: hardware and software. The hardware covers the central processing unit(s) (CPU), fast access memory to store programs and data and peripherial devices (such as disks and printers) which interact with the CPU.

An Operating System [13] is a set of programs used to assist in the operation and efficient use of the computer system resources. The operating system provides the user with tools which allow easy design, debbuging, coding and maintenance of programs, thereby permitting the efficient transformation of algorithms into code. In order to describe the activities of an operating system in a multiprogramming enviroment, i.e. when the computer excecutes several memory resident programs simultaneousely, measurements have to be made rather carefully.

The basic entity controlled by an operating system is the process. A process consists of an address space and both a hardware and a software context. A user (or job) is the accounting equivalent of the set of processes, that it and its subprocesses create. It is assumed, because of physical limitations, that each process is restricted to a limited address space, and to a limited amount of system resources. In such an enviroment processes have to compete to obtain control over the necessary resources. It is the task of the operating system to control the activities within the environment to assure a high level of system performance.

Modern finite element programs are usually fairly complicated in terms of their internal logic flow. The level of complication increases when such programs are implemented on a minicomputer with tight limitations on system resources. This implies the necessity of an in-depth analysis of the environment. Unfortunately, the user, and sometimes the program developer, has limited information regarding the operating system, and specifically the algorithms used to implement it. Computer science literature abounds with studies of the behavior of a variety of operating system algorithms. However, most such studies assume access to internal operating system code and data [14,15].

4. DESCRIPTION OF THE IGEL COMPUTER ENVIROMENT.

The performance meaurements were conducted at the Interactive Graphics Engineering Laboratory (IGEL) of the University of Arizona. The computer system in question is a DGC Eclipse S/230 minicomputer running under the AOS operating system. The system functions in a multiprocessing time sharing mode. The main memory of the Eclipse has 224K of 16 bits words. The external memory is in the form of a single 200 Megabyte disk drive. In addition, 2 floppy disk drives of 315 KB each, and one magnetic tape unit are supported by the operating system. Access to the system is possible through 12 terminals supporting both alphanumerics and graphics, as well as two dial-up ports and one 4800 baud line for communcation with other computer installations. The most popular programming language used is the DGC FORTRAN 5, which is a superset of standard ANSII FORTRAN.

The ECLIPSE S/230 [16] is a general purpose minicomputer with a hardware floating point unit. It's CPU has 8 accumulators (four 32 bit and four 16 bit regiters) and two 16 bit accumulators, which can be used as index registers. Its instruction set has been implemented using microprogramming. The machine makes extensive use of the stack concept, which is supported by the hardware. Because of the word length (16 bits) the user address space is limited to 32 KW. However, the ECLIPSE, with it's special memory management unit (MAP), can accomodate up to 256 KW. In the configuration with the MAP, it is the AOS operating system which is responsible for memory allocation to the users.

The AOS operating system [17] is a multiprogramming, time sharing operating system which allows up to 64 users working concurrently. AOS provides high level language support in addition to symbolic editors, binders, library routines and other software development tools. As of this point, only those aspects of AOS which are relevant to the present work will be addressed.

The Command Line Interpreter (CLI) is the basic interface between the operating system and the user working at the console. The CLI is used by the programmer to create and control job execution in the system. Similar facilities are found in other operating systems. It is possible to create CLI macrocommands, i.e. commands which consist of one or more simple CLI commands or macrocommands with associated dummy arguments. The recursive macrocommand feature has been used in the implementation of the system performance tools necessary to obtain the desired measurements.

To enable the operating system to manage physical memory, The memory is divided into 1 KW blocks - PAGES. Each process being executed is allocated a number of pages, which fulfill its ACTUAL demand. The operating system (AOS) keeps track of all activities within the various processes. From that it is clear that one of the most important parts of an operating system in a multiprogramming environment is that dedicated to memory management. Because of the limitation of the physical memory, it is advantageous to share some parts of the code between several users executing the same program simultaneousely. This does not allow the user more address space, but allows the system to have more RESIDENT users within the available physical memory. Users who can not get memory are SWAPPED ,i.e. their core image is stored on disk.

Another techique which is used mainly to fit large codes in limited physical memory is that of OVERLAYING. In this method part of the code is kept on disk, until neêded. Whenever necessary, it is read into a specified part of the core, and then executed.

An interesting example of dynamic memory allocation is given by the execution of FORTRAN programs under AOS. The DGC FORTRAN operates on the stack principle. One of the advantages of such a method of language implementation is that FORTRAN is recursive. On the other hand, excessive amounts of core are often used.

The AOS accounting functions record the details of user activity for charging purposes. It enters into the daily log file the amount of CPU time used by each process, the elapsed time (i.e. how long it has been in the system), and the number of 512 byte blocks transferred to and from the disk. It also keeps record of how many pages of physical memory the task has used and for how long (in page-milliseconds). Those measurements define the measurable characteristics of the process.

5. PERFORMANCE MEASUREMENT

The measurements presented here were made on a computer loaded with the operating system kernel and test only. This controlled environment was chosen to assure stable repetitive measurements.

It has been stated that each process has a summary of its activity recorded on the daily log file under its name. In addition, another set of data was recorded on a separate file by the CLI. The two sets of data contain:

1. Test start and stop clock time, as well as console name -recorded by the CLI.

2. Elapsed time (clock time between creation of the process and its termination), CPU time (in milliseconds), number of I/O transfers (in 512 byte blocks) and page residency time (in page-milliseconds)- recorded by AOS.

The tests conducted are defined in table 1. The first column consists of the name of the test. The second shows the name of the activity, or activities, involved and the number of concurrent executions.

Test limitations

Originally the plan was to measure the system behavior in an interactive environment. Later it was realized that auxilliary hardware was needed to simulate user activities on the console. Since this would have been beyond the available resources, it was decided to change the approach somewhat. However, in order to assure close similarity with the interactive environment, each executing process was assigned to one of consoles. It was also necessary to ensure that if two or more processes use the same code, that each has a different data base. A special starting program was created. Its task is to synchronize the execution of the CLI macros for all concurrent jobs. Each macro is associated with a specific console. All necessary input for the programs was prepared on disk input files. Such an arrangement allows for at least a partial simulation of the use of the character device driver.

The ECLIPSE system, even with 9 majorprocesses, maintained an acceptable response, as has been clearly demonstrated by a user who, by mistake, performed some text editing during the test.

Table 1. Test Profiles

name of the test	structure : name#name#...
BM1.TEST	BULKM*1
BM3.TEST	BULKM*3
BM7.TEST	BULKM*7
BM9.TEST	BULKM*9
DM1.TEST	DECOM*1
DM3.TEST	DECOM*3
DM7.TST	DECOM*7
DM9.TEST	DECOM*9
BM1DM1.TEST	BULKM*1 DECOM*1
BM2DM2.TEST	BULKM*2 DECOM*2
BM4DM4.TEST	BULKM*4 DECOM*4
BM1DM1F51F5LD1.	BULKM*1 DECOM*1 FORT*1 FORTLOAD*1
BM2DM2F52F5LD2.	BULKM*2 DECOM*2 FORT*2 FORTLOAD*2

6. CHOICE OF ACTIVITY PROFILES.

In order to simulate typical activities, and make meaningful measurements, it was decided to first measure the amount of resources taken by each process under ideal circumstances. This was accomplished by allowing only the desired process to reside in the computer and execute with minimum overhead. This was repeated several times for all processes described in section 2. Once these basic measurements were conducted, the effect of running several of these processes simultaneousely was considered. Each process was selected in turn, and several of the same process were initiated simultaneousely again on an otherwise unencumbered system. The number of parallel processes possible was restricted by the number of terminals available, which was ten. Next, certain combinations of processes were selected to simulate specific environments. For example, a mixture of pairs of mesh generation and decomposition operations represents a pure finite element computational environment, with no program development. One pair was executed, then more, until the number of available terminals were exhausted. The most varied environment was that involving mesh generation, decomposition, FORTRAN compilation and program linking. Of that only two sets could operate simultaneousely under the given restrictions.

7. TEST RESULTS

After each test run, the performance data were recorded on log files, as described above, by AOS and the CLI. These data were later processed to estimate the mean value and standard deviation of the measured characteristic.

Characteristics of test programs BULKM and DECOM.

Both programs use several files during their execution, and the code was longer than the user address space. This last characteristic forced the use of overlays. There was, however, a difference in the utilization of the overlays between the two programs. BULKM accesses its 13 overlays in an irregular pattern, whereas the DECOM program uses only 3 overlay areas, accessed sequentially. Whereas DECOM is essentially a batch program, which provides some output on the console, and is relatively compute bound, BULKM is interactive, and I/O bound to a great degree.

Characteristics of the Software environment

The Software enviroment characteristics is described by a memory utilization map for the AOS processes (without the kernel) and test processes. This data, presented in table 2, was obtained using the Process Enviroment Display Program (PED) which allows periodic snapshots of memory statistics. In the table, PMGR stands for "Program Manager", OP stands for "Operator", EXEC for "System Executive", XLPT for line printer handler, and CONSOLE.CLI is the test monitor. PED, DECOM and BULKM have been described previousely.

Table 2. Memory Utilization

System Processes :

name of the Process	Shared Code	Unshared Code
PMGR	0	12
OP	18	2
EXEC	0	21
XLPT	3	3
PED	0	9

Test Processes :

CONSOLE.CLI	18	3
DECOM	9	23
BULKM	23	9
FORT (9 overlays)	6 - 13	variable
FORTLOAD	12	13

Observations:

1. AOS uses a maximum of 101 KW of main memory (kernel uses 32 KW). A substantial part of the system routines are normally swapped out to disk, so that as a rule somewhere between 70 KW and 95 KW occupy fast memory.

2. DECOM and BULKM have different memory utilization patterns which should have an influence on their behavior.
3. During the test, the swapping activity was observed. It was found that up to 3 DECOM, or 4 BULKM processes can fit in memory without swapping.

8. DISCUSSION OF RESULTS

Figures 2 through 9 show plots of performance variables. Three environments are described. Environment 1 is a homogeneous environment with only one test, singly or multiply excecuted. Environment 2 is a mixed finite element environment with the process pair BULKM and DECOM, one or more pairs executing simultaneousely. Environment 3 is a mixed finite element and program development environment which includes a set of processes encompassing BULKM, DECOM, as well as a FORTRAN compilation (FORT) and a program linking process (FORTLOAD). Again one or two sets may execute at a time. Each type of environment is represented by a different curve.

Due to the scatter in measurements, each point on the plot is accompanied by a vertical line giving the minimum and maximum values measured. In addition, the population and standard deviation are supplied in brackets. The only exception was in the case of the third test, where the population was too small to provide acceptable satistical parameters [18].

Elapsed Time Measurements

The presented results are primarily concerned with the observed execution characteristics of the BULKM and DECOM programs. The overall behavior is similar for all plots. The

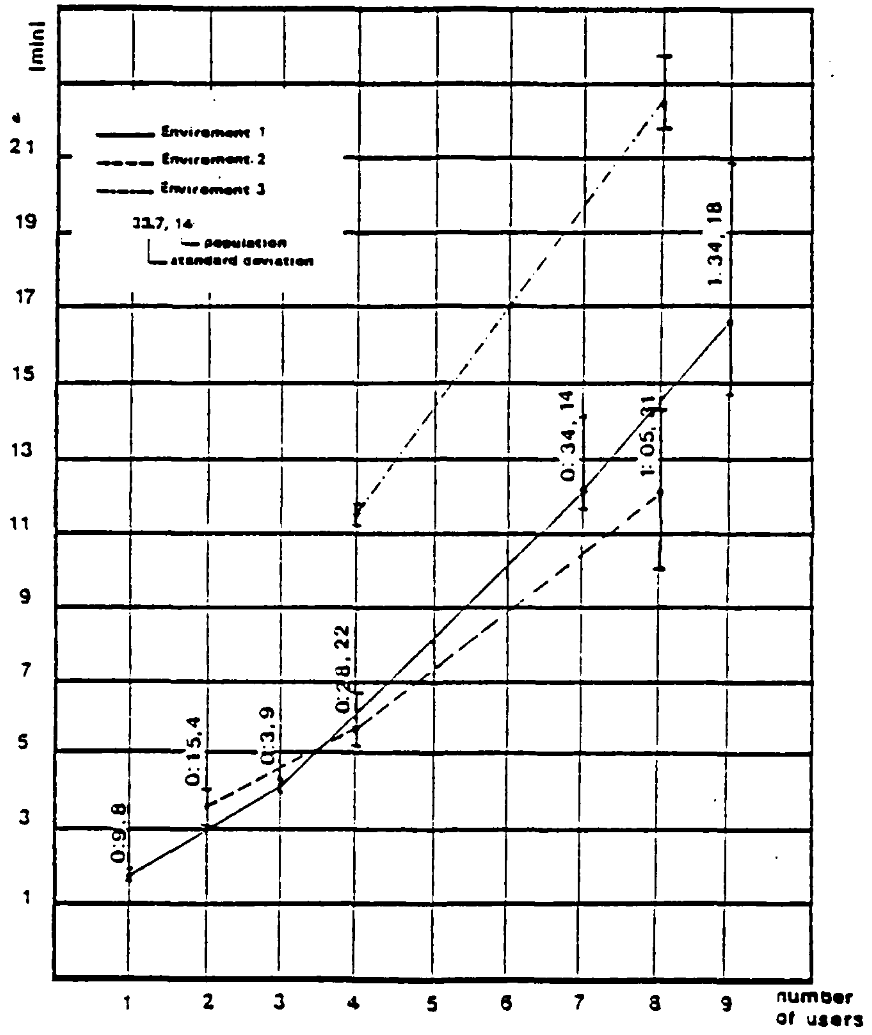

Figure 2. Elapsed Time BULKM

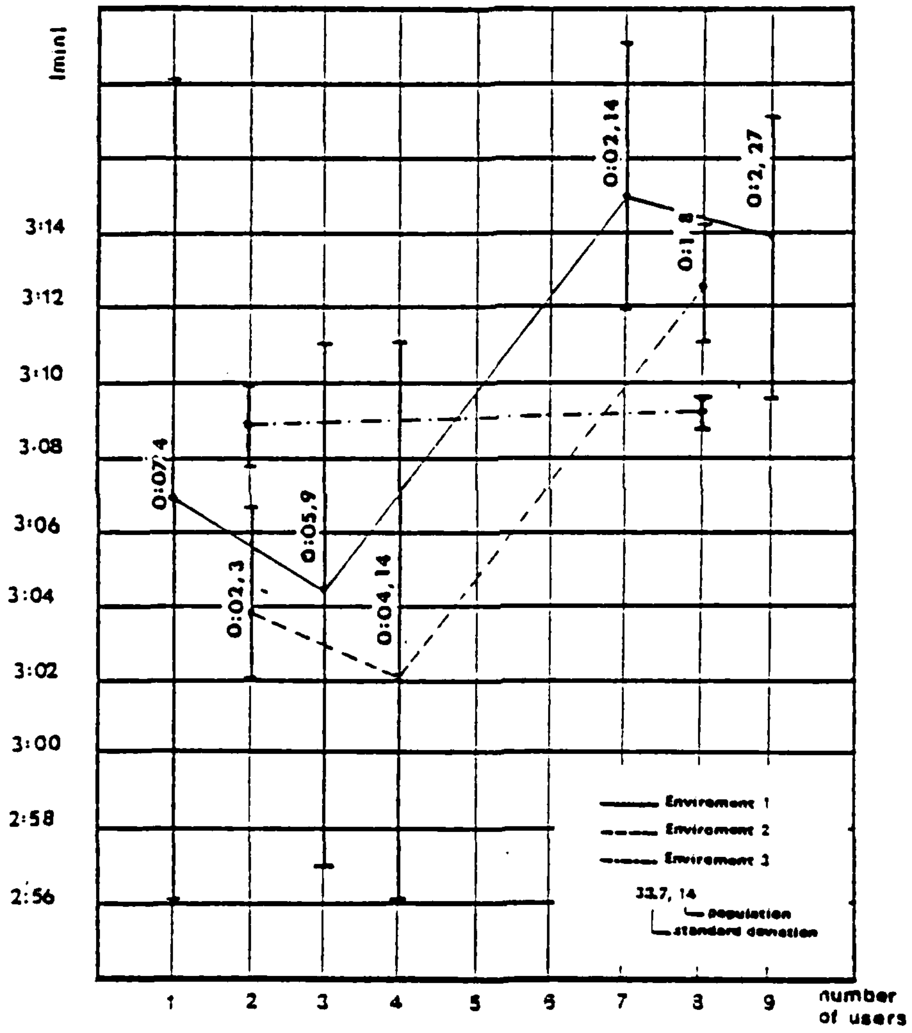

Figure 3. CPU Time DECOM

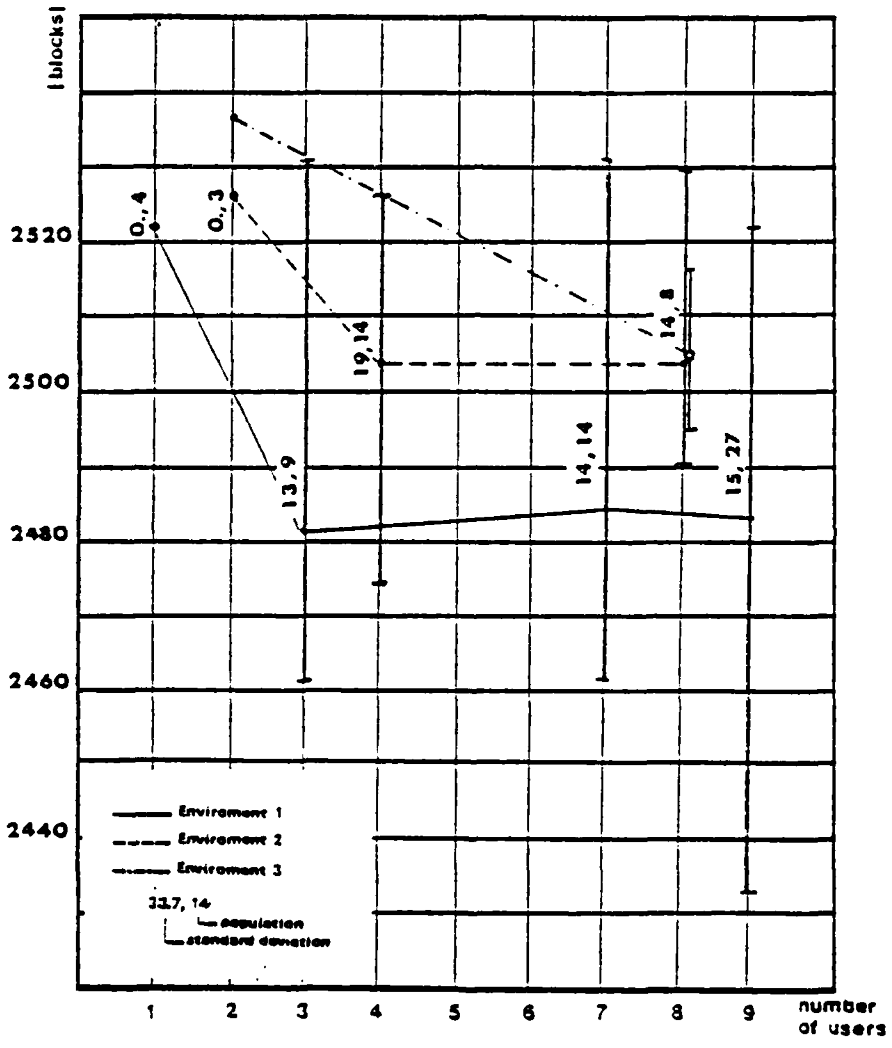

Figure 4. I/O Transfers DECOM

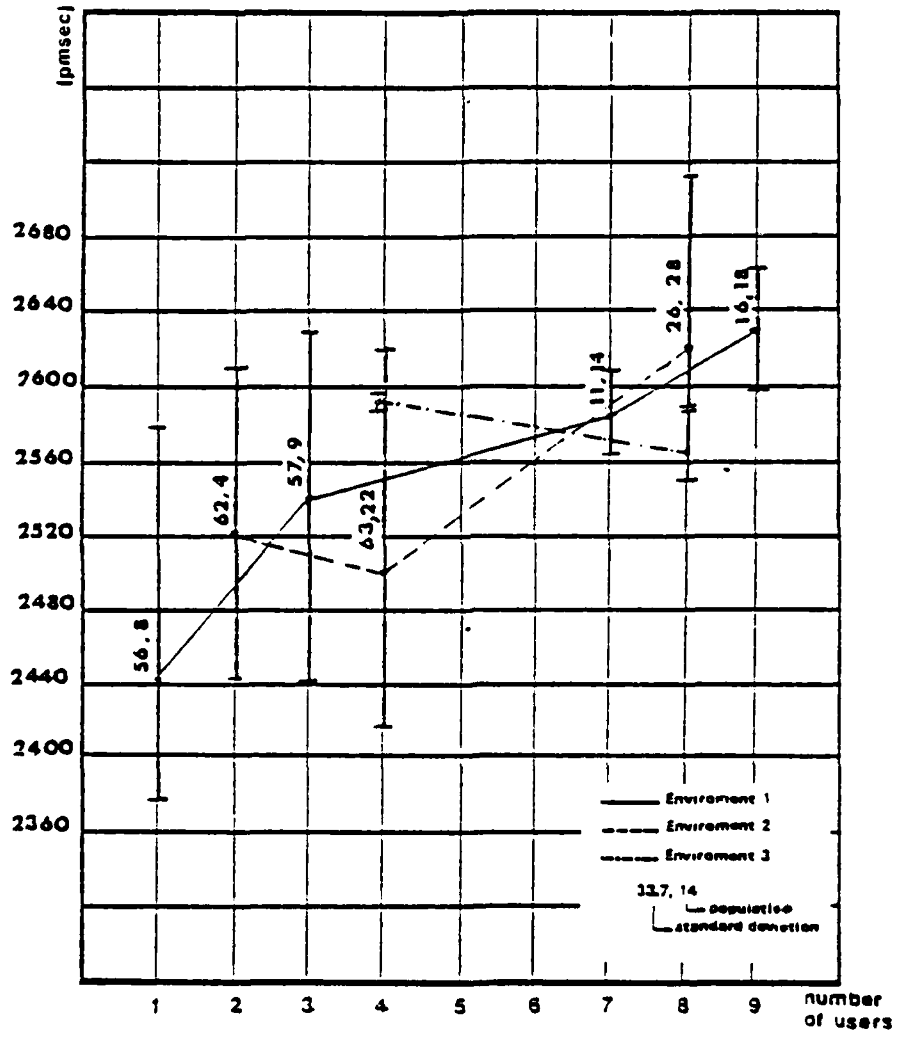

Figure 5. Page Residency Time BULKM

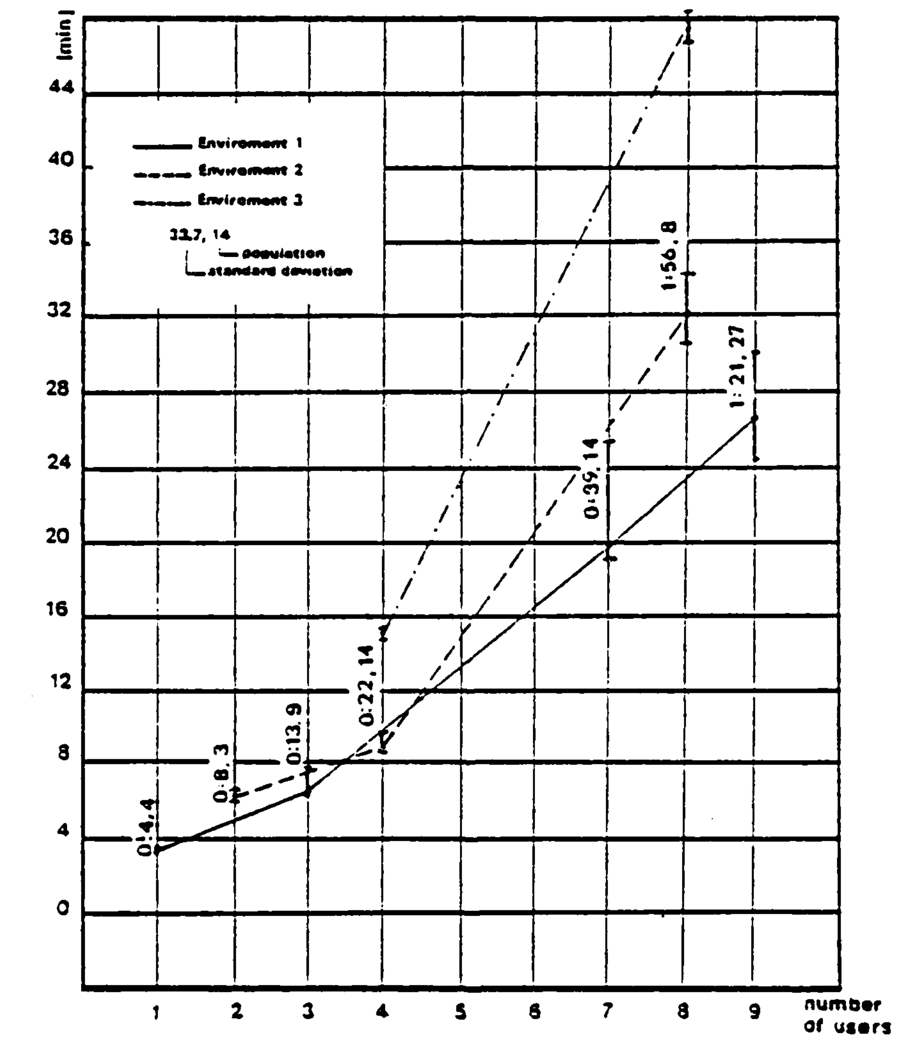

Figure 6. Elapsed Time DECOM

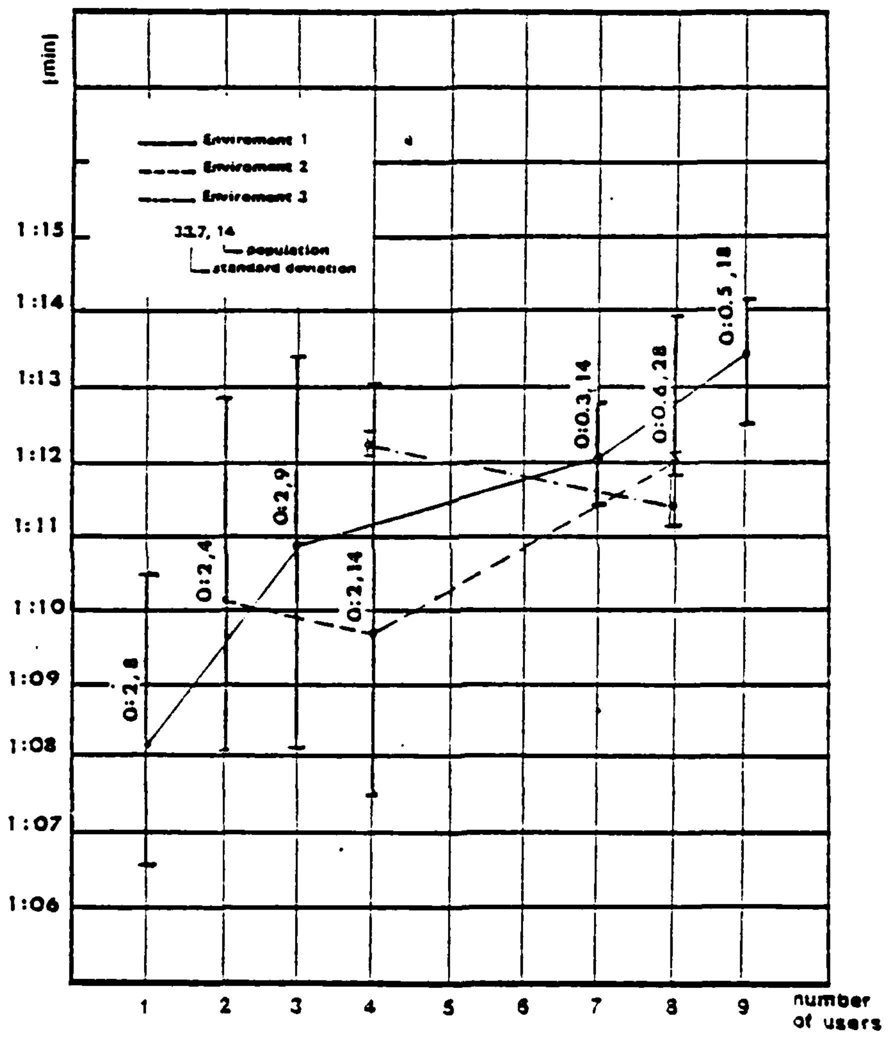

Figure 7. CPU Time BULKM

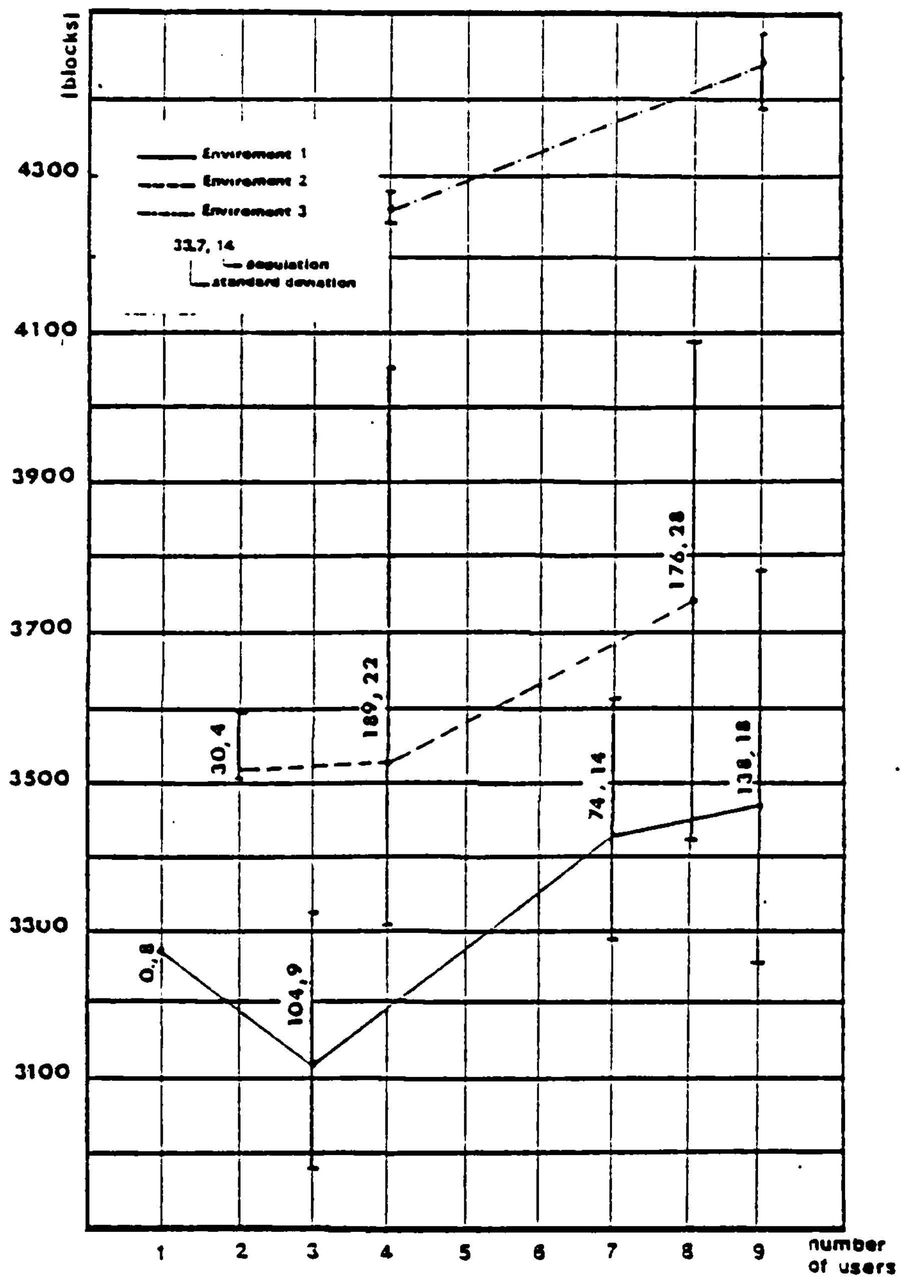

Figure 8. I/O Transfers BULKM

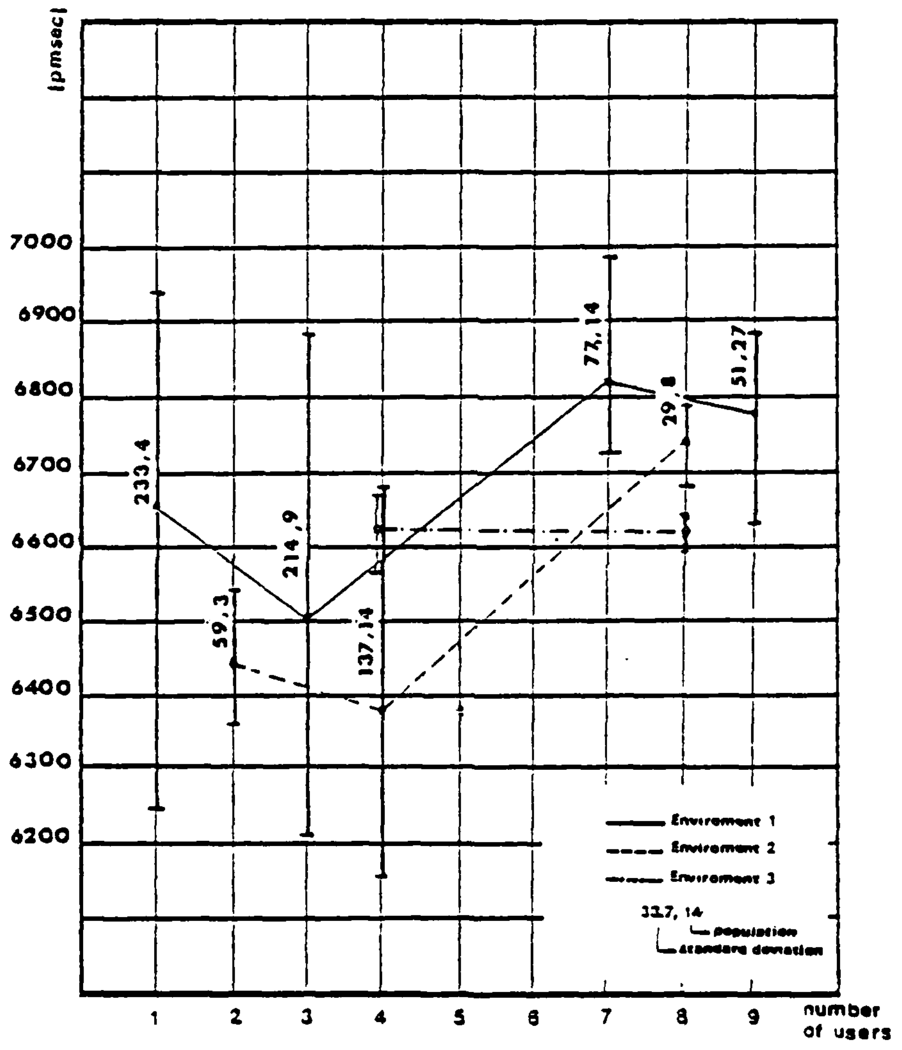

Figure 9. Page Residency Time DECOM

elapsed time grows nonlinearly with the number of users in the system. The main factors affecting the performance are the paging demand and overlay requests.

1. In order to evaluate the results of the analysis, a certain amount of discussion is needed. Let us consider an overlayed process which has relatively little code and a sequential mode of access to the overlays, such as DECOM. In a situation where a large number of identical users of the same type exist, there is a strong likelihood that a requested overlay is already present in physical memory and can therefore be shared between several users. there is a reduction of residence time for multiple processes, up until the core is saturated. This is illustrated by the different curves obtained for environments 1 and 2 in Figure 9. On the other hand, BULKM has a large number of randomly accessed overlays, and the total program size is such that it will take a corsiderable amount of physical memory to store all pages simultaneousely. In this case, the benefit is minimal. It can be observed that in the case of BULKM curves of environment 1 and 2 are the same, to within the error margin.
2. The CPU time variation can not be discussed without reference to I/O operations. It is assumed that the amount of CPU required to perform the required computation, and the CPU overhead associated with problem data I/O, is essentially constant. The only variable portion of the CPU must therefore be associated with paging due to overlay loading. A reduction is observed in the CPU time between the single user case and the three user case for program DECOM in environment 1. This can be explained by the fact that the single user has to load all overlays once,

whereas some overlay loading may be saved in a multiple user situation. The same is not true for BULKM due to the large number of overlays and the random access pattern. One expects to find a continuous increase in I/O and CPU values from the point of saturation of physical memory, which seems to occur here at an early stage. Another contributing factor to the nonlinear behavior of the CPU curve is the simple fact that in a time-sharing system the operating system uses more CPU time to reestablish user status with more users, whenever it gets its time slice.

3. The analysis of the page residency time in memory is based on the observations made in the last two sections. The page residency for program DECOM in environment 1 drops as the number of users increases from 1 to 3 for similar reasons as described above. An interesting point is how the behavior of DECOM influences that of BULKM in environment 2. Since BULKM performs more I/O operations than DECOM in the course of its execution. This results in DECOM being allocated proportionally more CPU, while BULKM is waiting for its I/O requests to be granted. This may explain the fact that in environment 2 DECOM uses less CPU time and page residency compared to environment 1.
4. It is not always easy to explain the behavior of complex dynamic environments, such as the one we are dealing with. The behavior of DECOM and BULKM in environment 3 is particularly interesting. It is important to note that the compilation and linking processes are heavily I/O bound. As a result, both programs reside longer within the computer, as expected, while their page memory residency, CPU time, and amount of I/O operations all decrease. This may be due to the fact that, since the program development

activities are I/O bound, DECOM and BULKM get more time slices, and are swapped less out of core. One should however be somewhat careful about reaching conclusions, since the number of samples were small in this case. This was necessary because of time limitations. For example the elapsed time for one execution of DECOM is about 30 min.

5. The results presented above will hopefully provide some measure of the performance of a typical minicomputer, and some possible explanations of observed behavior. Because of the heuristical nature of the process, some inaccuracy is expected. Other factors complicate matters. For example, the careful reader will observe that the CPU time recorded in the file for DECOM is longer than the elapsed time. The only possible explanation for that is a consistent, and hopefully minor, error in the system.

9. CONCLUSIONS

1. It is possible to conduct performance measurements to evaluate the effectiveness of a minicomputer in a finite element environment.
2. Altogether the 16 bit minicomputer used for these measurements behaved well under a variety of conditions representative of finite element computations, as well as a mix between that and program development activities.
3. It is important that the measured values are interpreted correctly and carefully, and that unnecessary generalizations are avoided.
4. It is expected that as more jobs are run simultaneousely on a time sharing system, the turn around time (elapsed

time) for the individual executions increase. It takes longer to perform the calculations, but the computer is performing more work during the given period.

5. Some jobs compliment others so that the presence of one job may actually result in the reduction of computer resources required to execute the other.
6. A higher residency time does not always mean higher resource utilization and higher computer costs.
7. Should the system run without overhead, and if the system treats all jobs equally, the turn around time should go up linearly with the number of jobs being executed. In practice a nonlinear overhead exists. The amount of this overhead, and nature of its nonlinearity, is a measure of the efficiency of the operating system.
8. It is possible to configure a system on the basis of measurements of program characteristics, in order to retain a certain throughput.
9. Some of the conclusions refer only to the specific computer system under consideration. Many of the methods used, however, and some general remarks may be extended to other installations [9,10,11,12].
10. It is possible to construct finite element benchmarks, such as those discussed here, in order to evaluate relative performance of minicomputer systems in a finite element environment.

ACKNOWLEDGEMENTS

The authors gratefully acknowledge the support of the Office of Naval Research under contract number N00014-75-C-0837, and that of the National Science Foundation under contract number ENG77-17013.

10. BIBLIOGRAPHY

1. FASTDRAW Interactive Graphics System, McDonnel-Douglas Automation Co., Document B1298083,1973.

2. Gingerich, M.M. Abe, Vinecore, R.L., Romans, G.J. and Ratihn, B.M., A Stand-alone Interactive Graphics Finite Element Modeling System. Sixth NASTRAN Users' Colloquium, NASA Conf. Pub. 2018, Oct. 77.

3. Kamel, H.A. and McCabe, M.W., GIFTS: Graphics Oriented Interactive Finite Element Time-Sharing System. Structural Mechanics Software Series, Vol I, Perrone N., Pilkey, W. (Eds.) U. P. of Virginia, 1977.

4. Buck, K.E., Bodisco, U.V., Winkler, K., SUPERNET, Data Generation for Finite Elements, BBC Report, Dept. ZKN/C, Mannheim, June 1977.

5. Shoppee, G.J.V., Jeanes, P.J. and Griffin, T.B., A Finite Element Modeling and Analysis Language for Engineering the Program FEMALE, Advances- Engineering Software, Vol. 1, No. 1, p 37-41, 1978.

6. Johansson, T., FEMGEN - A General Finite Element Mesh Generator. Application using ADINA, Bathe, K-J. (Ed.), April 77.

9. Mamrak, S.A.; Abrams, M.D.,"A taxonomy for valid test workload generation", Computer, vol. 12, number 12, 1979.

10. Agrawala, A.K.; Mohr, J.M.; Bryant, R.M.,"An approach to the workload characterization problem", Computer, vol. 9, number 6, 1976.

11. Ferrari, D.,"Workload Characterization and selection in computer performance measurements", Computer, Vol. 5, number 4, 1972.

12. Nolan, L.E.; Strauss, J.C., "Workload characterization for time-sharing system selection", Software - Practice and Experience, Vol. 4, 1974.

13. Shaw, Alan C.,"the logical Design of Operating Systems", Prentice-Hall, 1974.

14. Chu, W.W.; Opderbeck, H., "The page fault frequency replacement algorithm", Proceedings of the AFIPS 1972 Fall Joint Computer Conference.

15. Opderbeck, H.; Chu, W.W., "Performance of the page fault frequency replacement algorithm in multiprogramming enviroment", Proceedings of IFIP Congress 74.

16. "Programmer's reference manual, ECLIPSE-line Computers", DGC.

17. "Advanced Operating System (AOS), programmer's manual", DGC.

18. Kobayashi, H.,"Modeling and Analysis an Introduction to System Performance Evaluation Methodology", Addison-Wesley, 1978.

19. Grochow, J.M., "Utility functions for time-sharing system performance evaluation", Computer, Vol. 5, number 8, 1972.

ABSTRACT

The explosive rate of development of computer hardware has affected engineering practice, particularly the area of structural engineering and finite element analysis. Programs are now available, some on minicomputers, which perform pre- and postprocessing functions as well as analysis. Many small engineering groups, enticed by the availability of relatively inexpensive hardware, have acquired mini- and microcomputer systems with the intention of using them for day to day engineering computations. This paper attempts a study of the performance of a typical 16-bit minicomputer in a finite element environment. It is hoped that this study will give a realistic feel for the performance of such systems in a practical environment, and provide an example of how the performance of such systems may be measured.

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER N00014-75-C-0837, T.R. #6	2. GOVT ACCESSION NO. AD-A091648	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Performance of Minicomputers in Finite Element Analysis Pre and Post Processing		5. TYPE OF REPORT & PERIOD COVERED Technical 5/20/80
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Mai Tan, J. & Kamel, H.A.		8. CONTRACT OR GRANT NUMBER(s) N0014-75-c-0837
9. PERFORMING ORGANIZATION NAME AND ADDRESS University of Arizona Aerospace & Mechanical Engineering Dept. Tucson, AZ 85621		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS NR 064-531/12-17-75
11. CONTROLLING OFFICE NAME AND ADDRESS Dept. of the Navy, Office of Naval Research Structural Mechanics Program (Code 474) Arlington, VA 22217		12. REPORT DATE 7/29/80
		13. NUMBER OF PAGES 28
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Same as above		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release, distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

Presented at ASCE Spring Convention, Portland, Oregon April 14-18, 1980

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Finite Element Method, Minicomputers, Performance Measurements, Computational Algorithms.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The paper presents a study of the performance of a typical 16-bit minicomputer in a finite element environment. The study gives a realistic feel for the performance of such systems in a practical environment, and provides an example of how the performance of such systems may be measured.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF 014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)